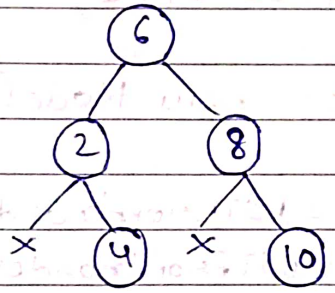
09/12/2023
Saturday

* **Construct Binary Search Tree from inorder -**

Given inorder array, we have to make BST using it.

I/P - inorder = [2, 4, 6, 8, 10]

O/P -
6
2 8
4 10
} level-order print of BST



Approach -

= Base case - (start > end) represents invalid array.

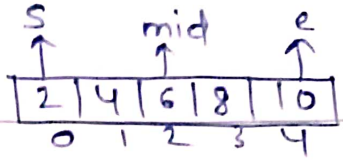
= The mid index of inorder array is the root node of BST.

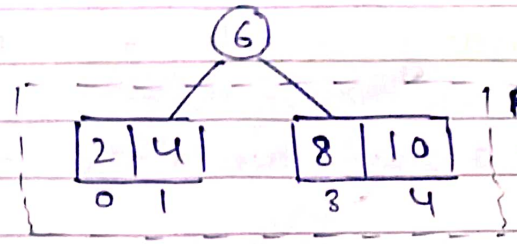
= left subarray, (s, e) = (s, mid - 1)

= right subarray, (s, e) = (mid + 1, end)

left and right subarray ko recursion solve kr dega.

For instance, inorder =





Recursion will solve this.

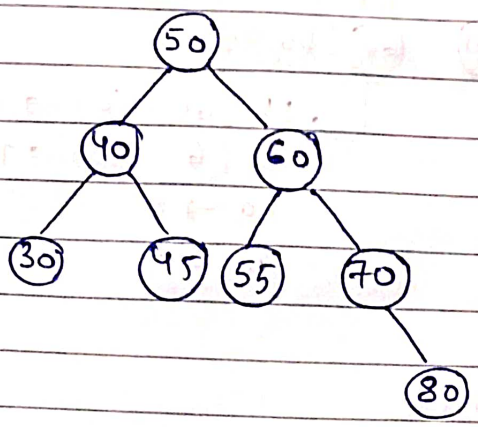
Code -

```
Node* BSTFromInorder (vector<int>&inorder, int s,
                      int e)
{
    if(s > e) return NULL;

    int mid = s+(e-s)/2;

    Node* root = new Node(inorder[mid]);

    root->left = BSTFromInorder(inorder, s, mid-1);
    root->right= BSTFromInorder(inorder, mid+1, e);

    return root;
}

int main() {
    vector<int>inorder{2, 4, 6, 8, 10};
    int start = 0;
    int end =inOrder.size()-1;
    Node* root = BSTFromInorder(inorder, start, end);
    levelOrderTraversal(root);
}
```

* Validate Binary Search Tree (leetcode-98)

Given root of a binary tree, determine if it is a valid BST.

I/P-



O/P- true

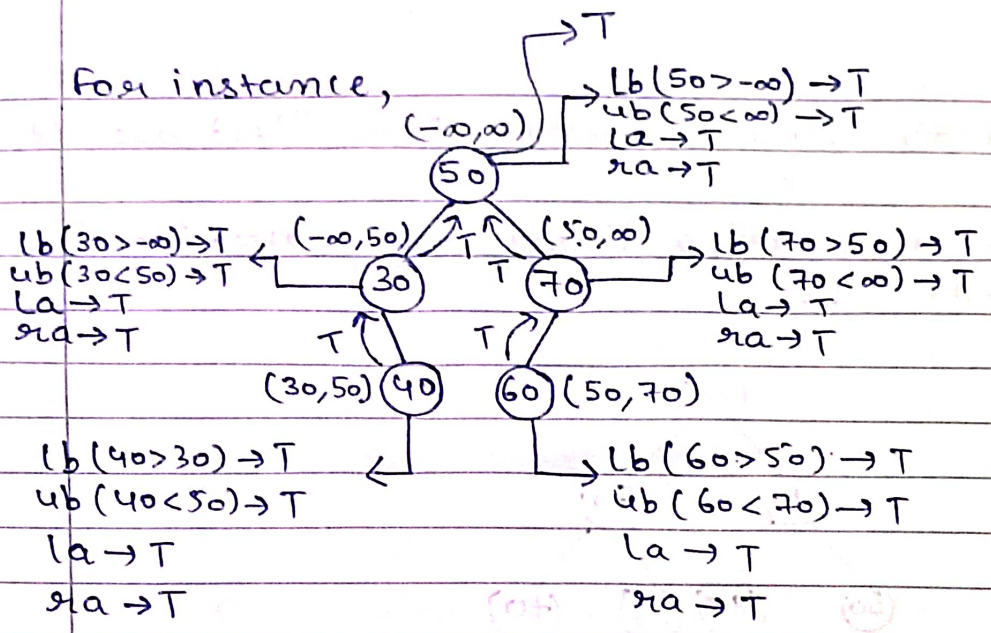
Approach-

At every node in binary tree, following 4 conditions must be true.-

(1) root -> val > lowerBound
(2) root -> val < upperBound
(3) left subtree should be BST.
(4) right subtree should be BST.

- Initially, at root node, the range i.e. (lowerBound, upperBound) is $(-\infty, \infty)$.
- Moving to the left of root node, the range changes to $(-\infty$, root -> val).
- Moving to the right of root node, the range changes to (root -> val, ∞).

For instance,



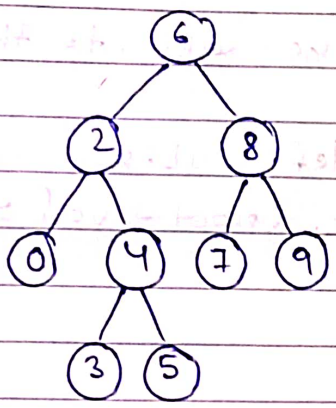
Code-

```
class Solution{
public:

bool solve(TreeNode* root, long long lowerBound,
           long long upperBound) {

    if(!root) return 1;

    bool cond1 = (root->val > lowerBound &&
                  root->val < upperBound);
    bool cond2 = (solve(root->left, lowerBound,
                        root->val) &&
                  solve(root->right, root->val,
                        upperBound));

    return cond1 && cond2;
  }
bool isValidBST(TreeNode* root){
     return solve(root, LONG_MIN, LONG_MAX);
   }
};
```

* **Lowest Common Ancestor of a Binary Search Tree (leetcode - 235)**

Given a BST, find the LCA node of two given nodes in the BST.

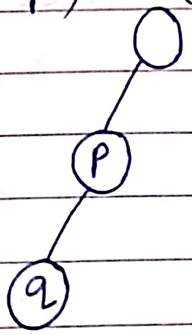
I/P – root = [6, 2, 8, 0, 4, 7, 9, null, null, 3, 5]
p = 2, q = 4



O/P – 2

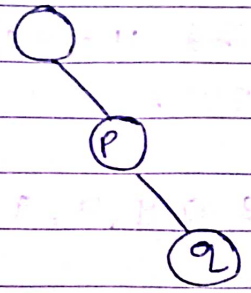
Approach –

Case 1 ⇒ When p and q are in left subtree i.e.
(root->val > p) && (root->val > q)



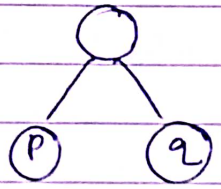
Make a recursive call to the root->left.

Case 2 - when p and q are in right subtree i.e. (root->val < p) && (root->val < q)



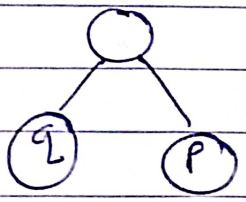
Make a recursive call to the root->right.

Case 3 - when p is in left subtree and q is in right subtree i.e. (root->val > p) && (root->val < q)



return root.

Case 4 - when p is in right subtree and q is in left subtree i.e. (root->val < p) && (root->val > q)



return root.

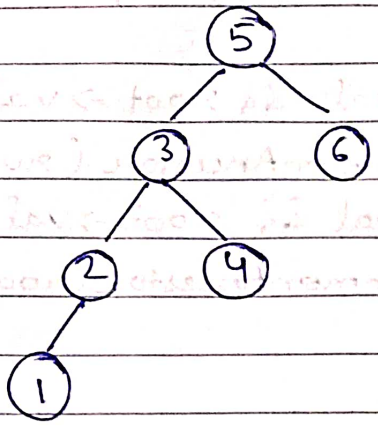
Code –

```
class Solution {
public:

TreeNode * lowestCommonAncestor (TreeNode* root,
        TreeNode * p, TreeNode* q)
{
    if (!root) return NULL;

 if (root -> val > p -> val && root -> val > q -> val)
   return lowestCommonAncestor (root -> left, p, q);
 if (root -> val < p -> val && root -> val < q -> val)
   return lowestCommonAncestor(root -> right, p, q);

   return root;
   }
};
```

* **kth smallest element in a BST (leetcode - 230)**

Given the root of a BST, and an integer k, return the kth smallest value of all the values of the nodes in a tree.

I/P - root = [5, 3, 6, 2, 4, null, null, 1], k = 3



O/P - 3

Approach -

Inorder traversal mei hum ascending order mei traverse kr rhe hote hai.

BST ki leftmost node pr jake return krte time k-- krte rhenge. Jis node pr k=0 ho jayega, vhi node → data kth smallest element hai.

Note - Is approach ka use krke hum BST mei nodes ki indexing kr skte hai.

Code -

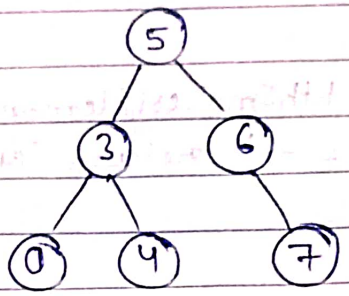
```
class Solution{
public:

int kthSmallestElement(TreeNode* root, int&k){

    if(!root) return -1;

    //L
    int leftAns = kthSmallestElement(root->left, k);
    if(leftAns != -1) return leftAns;

    //N
    k--;
    if(k==0) return root->val;

    //R
    int rightAns = kthSmallestElement(root->right, k);
    if(rightAns != -1) return rightAns;

    return -1;
    }
};
```

→ k ko pass by reference krna pdega so that k ka track rkh ske else return krte time jb previous node pe jayenge to k ki value accordingly node change ho jayengi.

* Two Sum IV - Input is a BST (leetcode-653)

Given the root of a BST and an integer k, return true if there exists two elements in the BST such that their sum equals to k, or false otherwise.

I/P →



root = [5, 3, 6, 0, 4, null, 7], k = 7

O/P → true

Approach –

= find inorder of the BST. It will be sorted.

= Use two pointers s and e, s = 0 & e = inorder.size() - 1

= if sum i.e. inorder[s] + inorder[e] = target sum, then return 1.

= else if sum > target sum, then e--.

= else if sum < target, then ~~sum~~ s++.

For the given input BST and k = 7,

inorder =

0	1	2	3	4	5
0	3	4	5	6	7
s	s	e	e	e	e

8 > 7
6 < 7
9 > 7
8 > 7
7 = 7 } return 1

Code –

```
class Solution{
public:
void BSTinorder(TreeNode* root, vector<int>&inorder){
    if(!root) return;

    BSTinorder(root->left,inorder);
    inorder.push_back(root->val);
    BSTinorder(root->right, inorder);
}

bool findTarget(TreeNode * root, int k) {
    vector<int>inorder;
    BSTinorder(root, inorder);

    int s=0;
    int e = inorder.size()-1;
    while(s<e){
        int sum= inorder[s]+inorder[e];
        if(sum==k) return 1;
        else if(sum>k) e--;
        else s++;
    }
    return 0;
}
};
```